ConneXions™

The Interoperability Report

January 1990 Volume 4, No. 1

ConneXions — The Interoperability Report tracks current and emerging standards and technologies within the computer and communications industry.

In this issue:

ConneXions is published monthly by Advanced Computing Environments, 480 San Antonio Road, Suite 100, Mountain View, California 94040, USA. Phone: 415-941-3399. Fax: 415-949-1779.

ISSN 0894-5926

From the Editor

We begin 1990 and Volume 4 of *ConneXions* with yet another installment in our series *Components of OSI*. As you will have discovered by now, the articles have been presented in no particular order. Maybe some day when all the pieces are in, we can assemble them all into a special issue, which would actually be a small book on OSI. Meanwhile, this month's topic is the OSI *Application Layer Structure* (ALS), presented here by Marshall Rose and largely taken from his book *The Open Book: A Practical Perspective on OSI.*

Security continues to be an important topic for today's computer networks. The MIT Project Athena has developed the *Kerberos* authentication system. The system, which is rapidly gaining popularity in the Internet community, is described by one of its designers, Jeff Schiller, in an article on page 10.

Security, or perhaps lack thereof, was also what sent Clifford Stoll on a spy-chasing mission through a maze of computer networks around the world a couple of years ago. The story is told in his recently published book entitled *The Cuckoo's Egg*, and is highly recommended for anyone involved with network or system administration. We asked Jon Postel to review the book. See page 15.

We are pleased to announce that Richard desJardins has joined Advanced Computing Environments as Vice President of program development. Dick has been involved in the OSI standardization process from the very beginning. Prior to joining ACE, he spent six years with Computer Technology Associates as chief engineer and a senior technical consultant in areas of distributed computing, network and protocol design, and space data systems. He has taught several courses on OSI, and will continue to do so for our Internetworking Tutorial series. As our certified in-house OSI expert, Dick will undoubtedly be called upon by yours truly to contribute to *ConneXions* from time to time.

Advanced Computing Environments will be offering an extensive set of courses in our 1990 *Internetworking Tutorials* program. There are over 20 tutorials to choose from, and courses will be offered at four different locations and dates throughout 1990. The first set of tutorials will take place in Los Angeles, California from January 29 through February 1, 1990. For a complete course program, call Advanced Computing Environments at 415-941-3399 or Fax us at 415-949-1779.

Components of OSI: The Application Layer Structure

by Marshall T. Rose, NYSERNet, Inc.

Introduction

As noted in articles appearing in earlier issues of *ConneXions*, the OSI upper-layer infrastructure is rich with functionality. It is hoped that this richness will allow applications designers and programmers to develop powerful applications for users of OSI networks. Let us briefly re-introduce these concepts:

- The OSI transport service provides full-duplex circuits, offering a similar capability to that of the eminently popular Transmission Control Protocol (TCP);
- The OSI session service builds on top of transport circuits by adding mechanisms that allow the user to *control* the dialogues between applications that are specific to a particular application task, e.g., accessing a file or delivering a mail message; and,
- The OSI presentation service builds on top of session dialogues by adding mechanisms that allow the user to define the *structure* of data that are exchanged between applications, e.g., the structure of the records in a file or the body parts contained in a mail message.

The OSI application layer builds on these services provided by the layers beneath.

Application Contexts

To begin, the OSI application layer is divided into *application service elements*, each with a particular function. To define an application protocol, one combines different service elements and decides the rules for the ways in which these service elements interact with themselves and the underlying OSI services (as seen through the presentation service). For example, all applications ultimately establish an association with a peer. The *Association Control Service Element* (ACSE) is responsible for this task. Hence:

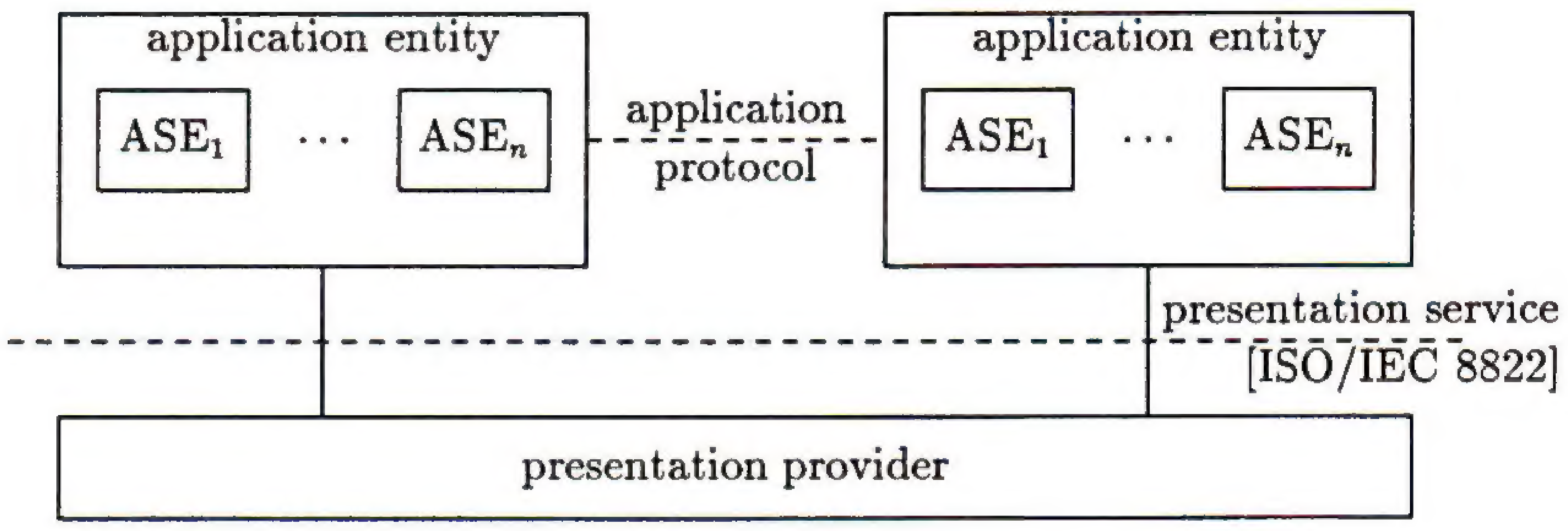
- All OSI applications contain the ACSE in order to perform association establishment and release.
- All OSI applications contain definitions stating how the ACSE will be used (e.g., which other service elements might invoke the services of the ACSE);
- And, only the ACSE is allowed to invoke the connection establishment and release services of the presentation layer below.

The combination of all service elements that comprise the application, along with the relationship between those service elements and the underlying OSI services form an *application context.*

It should be noted that in addition to making application protocols more tractable, the use of application service elements promote reuse of application layer facilities. As with the OSI upper-layer infrastructure as a whole, from an implementation viewpoint this effect can provide substantive leverage.

Application Entities

In an OSI environment, *application processes* are "things" which execute in the network and provide (presumably) useful service to end-users. The OSI communications aspects of these processes are termed *application entities*. Hence, an application entity is a "thing" composed of one or more *application service elements* (ASEs) which provide the particular application protocol defined by the application context. Conceptually, we might view these relationships as:



Note that each of the peer application entities is composed of precisely the same ASEs. Further, each ASE talks only with its peer in the remote application entity. To make sure that the application protocol data units are always delivered to the correct ASE peer, each ASE defines a *presentation context* to be used over the association. When data is given to the presentation layer, the data is marked with the appropriate presentation context so as to ensure correct delivery on the remote system.

There are several application service elements that may be part of an application context. Although several are progressing through the standards process, at this time only three of the commonly used service elements have reached consensus:

- Association control;
- Remote operations; and,
- Reliable transfer.

Let's now take a look at these three service elements in a little more detail. As these are examined, other issues relating to the OSI application layer structure will be introduced.

Association Control Service Element

The *Association Control Service Element* (ACSE) is responsible for *association* establishment and release. An association is a binding between two entities, which are referred to as the *initiator* and the *responder*. The primary task of the ACSE is to bind one application entity to another. This binding results in an association that is supported by an underlying presentation connection.

Note that although binding implies a client/server model for establishing the association, this needn't have the same relationship for the services provided. It is perfectly reasonable for an initiator to bind to a responder for the purpose of letting the responder request actions to be performed by the initiator. This provider/consumer model is contrary to the familiar client/server model in which the client connects to the server, and then the client proceeds to request actions to be performed by the server.

The binding process is really two-step: First, the initiator determines which service it requires, and asks to have this service mapped onto the application entities running in the network.

continued on next page

The OSI Application Layer Structure *(continued)*

Second, based on the initiator's communications requirements, an association will be bound to one of those entities that becomes the responder.

Use of the OSI Directory

The first mapping is performed by OSI Directory Services, which acts as a "nameservice" in order to establish a binding. The application provides the *application entity title* of the desired service, which is registered in the OSI Directory as an information object. A Directory "read" operation is performed to retrieve the "presentationAddress" attribute of the object in question. In simplest terms, the OSI Directory acts as rendezvous point for the initiator and responder. When a service is to be offered in the network, the application which provides that service registers itself and its location(s) in the OSI Directory. Later on, when another process wishes to use that service, the OSI Directory is consulted to retrieve the appropriate addressing information.

The second mapping is performed by the initiator's transport provider, and is not germane to the discussion at hand. However, it is instructive to consider the causality of events which occur in between the time an application decides to establish an association and the initiator's transport provider is invoked.

Suppose that the application entity contains two ASEs: the ACSE and the FTAM ASE [FTAM, the OSI file service, will be described in a future issue of *ConneXions —Ed.*] When an association to the file service is to be established, the FTAM ASE generates an data object to initialize the remote FTAM ASE. This object is termed an `F-INITIALIZE-request` FPDU (*file protocol data unit*), and is defined in the abstract syntax for the FTAM protocol. The FTAM ASE then invokes the association establishment service of the ACSE, passing along this data. Thus, the FTAM ASE must request a presentation context for this syntax, along with a presentation context for the ACSE. Since each file in FTAM is defined in terms of a document type which contains structuring definitions, the FTAM ASE might also request a presentation context for each document type that it wishes to exchange over the duration of the association.

In this example, there are three document types specified. Presentation context identifier 1 is used for the FTAM PCI, contexts 3, 5, and 7 are used for the abstract syntaxes for the three document types, and context 9 is used for the ACSE PCI. Hence, the FPDU generated looks something like this:

```
{
  service-class {
    management-class, transfer-class,
    transfer-and-management-class
  },
  functional-units {
    read, write, limited-file-management,
    enhanced-file-management, grouping
  },
  attribute-groups { storage },
  ftam-quality-of-service no-recovery,
  contents-type-list {
    1.0.8571.5.3,      -- FTAM-3 document
    1.0.8571.5.1,      -- FTAM-1 document
    1.3.9999.1.5.9     -- NBS-9 document
  },
  initiator-identity "cheetah"
}
```

(This is, of course, a conceptualization—real implementations of OSI will use compact binary encodings for these data structures!)

Upon receiving the connection establishment request, the local ACSE generates a data object to initialize the remote ACSE. This object is termed AARQ APDU (*application protocol data unit*), and, as you might expect, the APDU is passed as user-data when the ACSE invokes the connection establishment service of the presentation layer. From the perspective of the ACSE, which does not need to understand the data types defined in the FTAM abstract syntax, the APDU looks something like this:

```
{
  application-context-name 1.0.8571.1.1     -- iso ftam
    user-information {
     {
        indirect-reference 1,               -- Indicates FTAM PCI
        encoding {
           single-ASN1-type {
              [3] '0370'H,
              [4] '053700'H,
              [5] '0580'H,
              [6] '00'H,
              [7] {
                  [APPLICATION 14] '28c27b0503'H,
                  [APPLICATION 14] '28c27b0501'H,
                  [APPLICATION 14] '2bce0f010509'H
              },
              [APPLICATION 22] "cheetah"
        }
     }
  }
}
```

Upon receiving the connection establishment request, the local presentation entity generates a data object to initialize the remote presentation entity. This object is termed a CP-type PPDU (presentation protocol data unit). The presentation service then serializes the PPDU, and passes the result, a string of 193 octets, as user-data when the presentation entity invokes the connection establishment service of the session layer. From the perspective of the presentation service, the PPDU looks something like the example on the following page.

Finally, upon receiving the connection establishment request, the local session entity generates a data object to initialize the remote session entity, and establishes a transport connection. If the transport connection is established, the data object is sent to the remote entity.

Reliable Transfer Service Element

The *Reliable Transfer Service Element* (RTSE) is responsible for bulk-mode transfers. Although the functionality of the session service provides for explicit checkpointing and connection recovery, many applications, whilst desiring the service (or a portion thereof), may find it daunting to manipulate these services directly. The RTSE is intended to hide the complexity of the session and presentation services to provide a simple transfer facility.

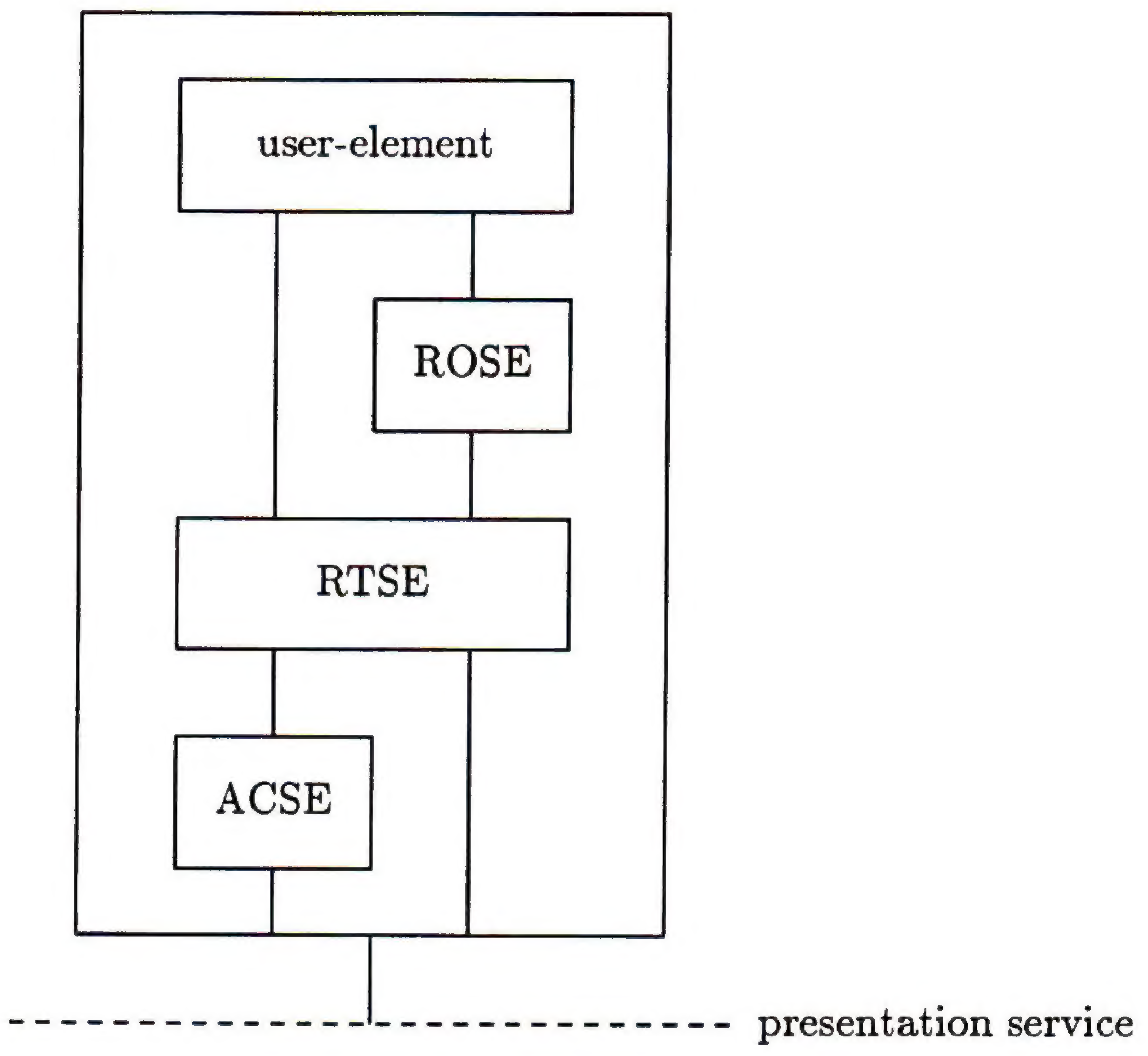
continued on next page

The OSI Application Layer Structure *(continued)*

```
{
  mode {
      normal-mode
  },
  normal-mode {
      context-list {
           {                    - ftam pci abstract syntax
              identifier 1,
              abstract-syntax 1.0.8571.2.1,
              transfer-syntax-list { 2.1.1 }
           },
           {                    - FTAM-3 abstract syntax
              identifier 3,
              abstract-syntax 1.0.8571.2.4,
              transfer-syntax-list { 2.1.1 }
           },
           {                    - FTAM-1 abstract syntax
              identifier 5,
              abstract-syntax 1.0.8571.2.3,
              transfer-syntax-list { 2.1.1 }
           },
           {                    - NBS-9 abstract syntax
              identifier 7,
              abstract-syntax 1.3.999.1.2.2,
              transfer-syntax-list { 2.1.1 }
           },
           {                    - acse pci abstract syntax
              identifier 9,
              abstract-syntax 2.2.1.0.1,
              transfer-syntax-list { 2.1.1 }
           }
        },
        user-data {
           complex {
              {
                 identifier 9,          -- Indicates ACSE PCI
                 presentation-data-values {
                    single-ASN1-type {
                       [1] { 1.0.8571.1.1 },
                       [30] {
                          {
                             indirect-reference 1,
                             encoding {
                               single-ASN1-type {
                                   [3] '0370'H,
                                   [4] '053700'H,
                                   [5] '0580'H,
                                   [6] '00'H,
                                   [7] {
                                       [APPLICATION 14] '28c27b0503'H,
                                       [APPLICATION 14] '28c27b0501'H,
                                       [APPLICATION 14] '2bce0f010509'H
                                   },
                                   [APPLICATION 22] "cheetah"
                             }
                          }
                       }
                    }
                 }
              }
           }
        }
     }
   }
}
```

Example CP-PPDU

The RTSE itself can be used by other service elements. For example, in Message Handling Systems (X.400), the Remote Operations Service Element often uses the RTSE, as the information exchanged may contain large electronic mail messages. Further, the RTSE can use other service elements, such as the ACSE. The idea is that the RTSE provides an additional level of abstraction in which the rich functionality of the underlying services are provided without the additional complexity of knowing how to use them in detail. Thus, an application might be configured like this:



In this example, the application context contains four service elements:

- The ACSE, to manage associations;
- The ROSE, to manage request/reply interactions;
- The RTSE, to provide for bulk-data transfer; and,
- A *user-element*, which is responsible for orchestrating the application entity's actions.

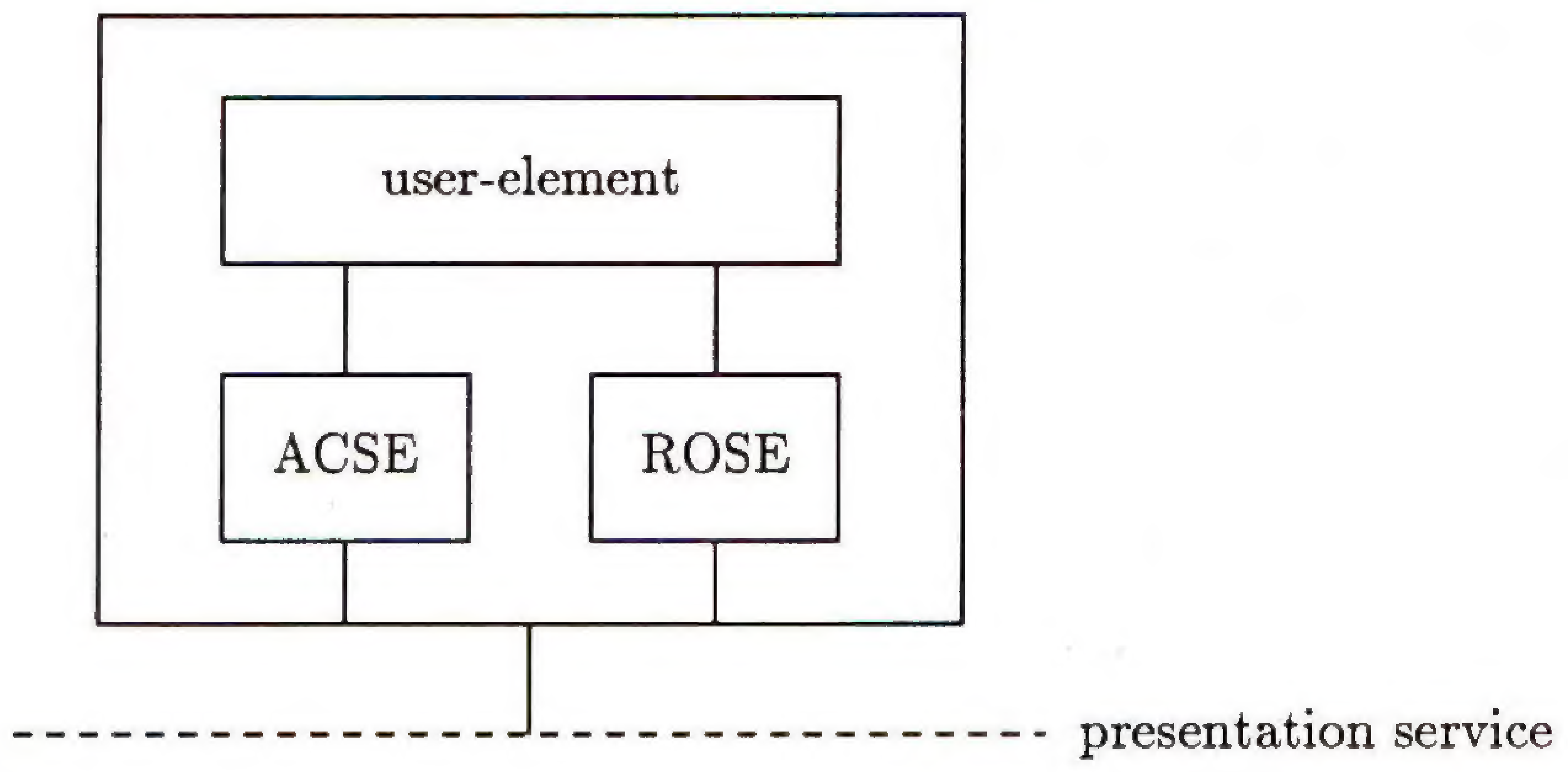
This has two implications:

- The user-element uses RTSE services to manage the association, which, in turn, use ACSE services; and,
- The ROSE uses RTSE services to transfer data which, in turn, use the presentation service.

continued on next page

The OSI Application Layer Structure *(continued)*

In contrast, an alternative configuration might be:



in which:

- The user-element uses ACSE services directly for association management; and,
- The ROSE uses the presentation service for data transfer.

Note that the same configuration must be present in both the application entities using an association. It is not possible for communication to occur if one AE contains an active RTSE and the other does not.

Remote Operations Service Element

The *Remote Operations Service Element* (ROSE) is responsible for request/reply *interactions*. An operation is *invoked* by an application process. In response, its peer returns one of three outcomes: a *result*, if the operation succeeded; an error, if the operation failed; or, a *rejection*, if the operation was not performed (e.g., due to network failure).

A surprising large number of applications use the ROSE. For example, the ROSE is used by message handling systems, directory services, network management, and remote database access. The reason for this is that the ROSE can easily support a *remote procedure call* (RPC) facility, which is what many distributed applications are built on. Many feel that the generality of the ROSE and its ability to support a wide range of loosely coupled systems may very well be a key factor in the overall success of OSI.

In addition to support for RPC, the ROSE also supports the notion of *linked operations* (sometimes termed a *callback* or *remote upcall*). The idea is simple: when an invocation is made, one of its arguments contains the name of an operation to be invoked as a part of the original invocation. This is useful if the performer of the original invocation requires additional information from time to time from the invoker. For example, one could imagine an operation called `Traverse` that took two arguments: the name of a directory containing the user's files, and the name of an operation to invoke for each of the files in that directory. To construct the "print directory" command, which prints each of the user's file on a laser printer, one need only invoke `Traverse` with the name of the directory along with the operation `PrintFile`.

If instead, the user wanted to list each file to the screen, the second argument would be `ListFile` instead. In both cases, the `Traverse` operation reads the user's directory and for each file found, it simply invokes the second argument with the name of the file.

It should be noted that the current OSI method for providing remote operations is based on a connection-oriented (CO-mode) model, while currently deployed non-OSI systems have successfully demonstrated the usefulness of a connectionless (CL-mode) approach. Fortunately, work on the CL-mode OSI upper-layers has completed. Unfortunately, work on CL-mode support for the ROSE is currently stalled in committee for non-technical (i.e., political) reasons.

For more information

Each of the commonly used service elements are defined in two parts: a service definition and a protocol specification. The ISO numbers are:

ASE	service	protocol
ACSE	8649	8650
RTSE	9066-1	9066-2
ROSE	9072-1	9072-2

For information on application service elements specific to a particular application, consult the standards which define that application. For example, the OSI Directory defines two other application service elements (along with the correspondent application contexts). These are defined in the OSI Directory standard.

For general information on the OSI application layer structure, the most recent document on the subject the author has seen is:

"Information Processing Systems—Open Systems Interconnection—Application Layer Structure." ISO/IEC Draft Proposal 9534, March 1987.

...which is admittedly rather dated. Fortunately, the contents of this draft proposal are also unreadable and bear little, if any, relationship to standardized OSI applications. It is beyond the scope of this paper to speculate why the standards community is able to standardize applications but is unable to standardize the model used for describing applications.

[This article is largely taken from Marshall Rose's *The Open Book: A Practical Perspective on OSI*, ISBN 0-13-643016-3. © 1989 by Prentice-Hall, Used with permission. —*Ed.*]

MARSHALL T. ROSE is Senior Scientist at NYSERNet, Inc., where he works on OSI protocols and network management. He is the principal implementor of the ISO Development Environment (ISODE), an openly available implementation of the upper layers of the OSI protocol suite. He is the author of *The Open Book: A Practical Perspective on OSI*, a professional text discussing the OSI in both theory and practice. Rose received the Ph.D. degree in Information and Computer Science from the University of California, Irvine, in 1984. His subscriptions to *The Atlantic* and *Rolling Stone Magazine* are in good standing.

Kerberos: Network Authentication for Today's Open Networks

by Jeffrey I. Schiller, Massachusetts Institute of Technology

Motivation

The predominant authentication mechanisms on the Internet today are passwords (via Telnet) and The Berkeley trusted hosts "R" commands (*rlogin, rsh, rcp,* etc.). Although adequate when first introduced, such mechanisms are rapidly becoming obsolete in today's Internet. The primary factors that lead to this conclusion are the explosive growth of the network and the increasing number of computer and network literate "crackers" which have surfaced.

The primary vulnerability of the "R" commands is the trusting of one host by another. The presence and use of these commands does not a priori represent a security vulnerability, but instead facilitates cracking activities by making it easy for a cracker, having broken into one host, to break into other hosts which trust it. The cracker can then repeat the process, hopping from one host to another as long as a trust relationship exists. What makes this particularly problematical is that it is often not possible for system administrators to know which hosts their own system trusts, for individual users can establish a list of hosts which are trusted to login to their account. When a host trusts another host, in fact, it is trusting all hosts which that host trusts. It is quite hard, sometimes, to determine where the trust stops. In most environments the perimeter of trusted hosts may be quite large (by perimeter, in this case, I mean the size of the set of hosts which trust each other).

The use of passwords over the network is also cause for concern. Unlike its predecessor networks, today's Internet is composed of many broadcast based network technologies. These technologies make the interception of network traffic by unauthorized third parties easier then ever. Although this implies that all information carried over such broadcast networks is in fact vulnerable to being intercepted, passwords are particularly valuable in that they are clear targets of would be network crackers.

What Kerberos Does

Realizing these problems and vulnerabilities MIT, through *Project Athena*, developed and deployed the *Kerberos* authentication system. Kerberos is an encryption based authentication system that allows a network user to authenticate him/herself to network service providers without the use of a trusted host relationship, or by transmitting a password in plain text over the network. In fact, an entity capturing all traffic on the network during a Kerberos authenticated transaction will not receive any information that will then allow it to masquerade as the user involved in the transaction.

How Kerberos Works

A user at MIT's Project Athena is greeted by a workstation displaying what appears to be a standard UNIX login prompt. However, whereas in a traditional UNIX system the typed name is looked up in a password file and the typed password compared against an encrypted password in that file, Athena uses the typed name and password to obtain a *ticket* for network services. This ticket comes with a randomly generated *session key*. Having possession of the ticket and knowing the session key, a user (or a program working on behalf of the user) can generate an *authenticator* which is then used to prove identity to other network services.

The key to the security of Kerberos is that the ticket and session key are provided to the users' workstation encrypted in a *Data Encryption Standard* (DES) key which is a function of their typed password. In this fashion no information travels across the network in the clear.

Tickets have limited lifetimes. This is important, for it limits the length of time that mischief may be done if a ticket and its matching session key are inadvertently disclosed. This is particularly important in the Project Athena environment where most workstations are "public," which is to say that they are located in public facilities and may be used by any of the 10,000 members of the Project's user community. If a workstation is left unattended while tickets are stored in it, the information that an unauthorized person can obtain has a limited usefulness. In particular the original user's password is in no way stored on the workstation and is not vulnerable to disclosure.

The basic transactions that make up the Kerberos protocol are as follows:

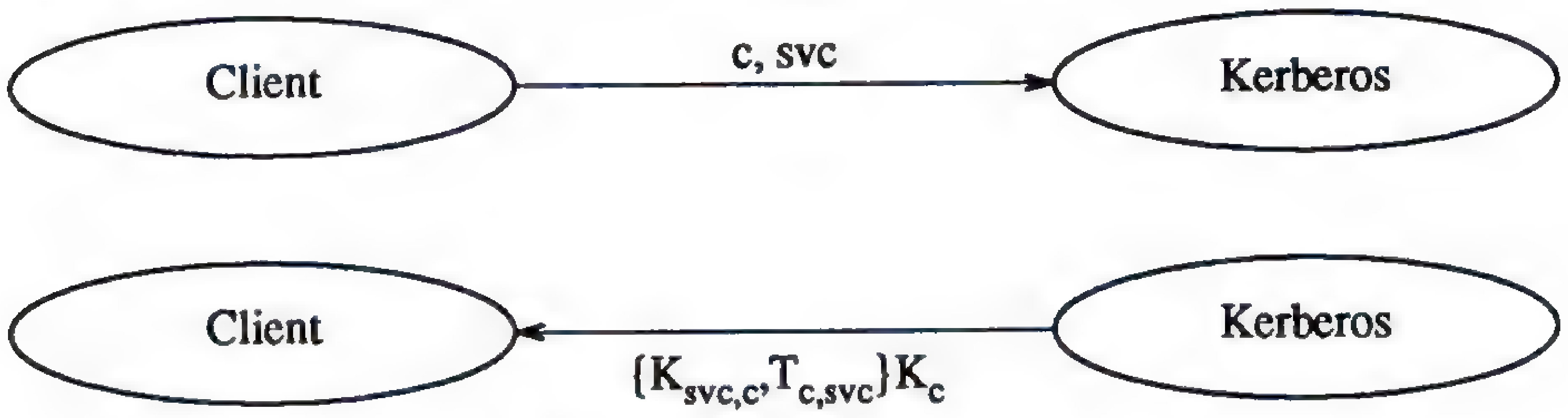


Figure 1: Getting a ticket

First a ticket is required. This is obtained (Figure 1) by sending a *request for ticket message* from the client workstation to the Kerberos server requesting a ticket for the service (*svc* in this case). The name of the user (*c* in this case) is supplied unencrypted. The Kerberos server responds with the Ticket and the corresponding session key encrypted (represented above by enclosing it in braces) in the private key of "c" (K_c). The ticket itself is a piece of data that contains a timestamp, lifetime, identification of "c," the name of "svc," Internet Address (addr) of the client workstation, and the session key ($K_{svc,c}$), all encrypted in a key known only to the service "svc" and the Kerberos server (K_{svc}).

$$T_{svc,c} = \{svc, c, addr, timestamp, life, K_{svc,c}\}K_{svc}$$

Figure 2: A Kerberos ticket

Once armed with the ticket and session key (obtained by decrypting the message sent in Figure 1 by the Kerberos server) a service request is sent by the client to the service in Figure 3, below:

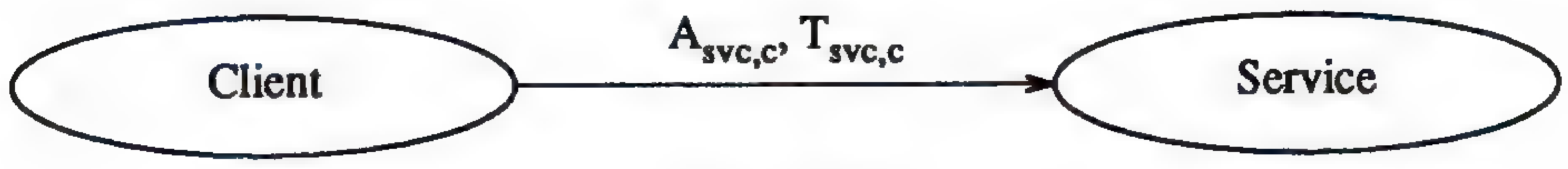


Figure 3: Requesting a Service

continued on next page

Kerberos: Network Authentication *(continued)*

The ticket itself is sent to the service along with an *authenticator*. The authenticator is constructed given the session key of the service and information known to the client. The authenticator is in fact the identification of the client user, "c," the IP address of the client workstation and timestamp, all encrypted in the session key of the ticket (Figure 4).

$$A_{svc,c} = \{c, addr, timestamp\}K_{svc,c}$$

Figure 4: A Kerberos Authenticator

Upon receiving the authenticator and ticket, the service verifies them by first decrypting the ticket using the key that it shares with the Kerberos server (K_{svc}). Once the ticket itself is decrypted the service then has access to the user's session key which it then uses to decrypt the authenticator. The service then compares the user's identity between the authenticator and ticket, verifies the time-stamps and lifetimes of both the ticket and authenticator, and finally verifies if the supplied client workstation address match between the authenticator, ticket and the received datagram source address. (Some care must be taken with multi-homed workstations).

The above description provides the basic core of the protocol. However if followed exactly as shown, the client user would have to enter his password (to generate his private key) every time a ticket for a new service was required, or his private key would need to be stored on the workstation. Both of these are situations that we wish to avoid. This is accomplished via an indirection called the *Ticket Granting Service* (TGS). The TGS runs on the same system as the Kerberos server and requires direct access to the Kerberos database. To use the TGS, first a ticket for it is obtain as outlined above. The ticket for the TGS can then be used to obtain other tickets from the TGS. These tickets are then returned encrypted in the session key of the TGS ticket rather then the user's password. In this way the only thing that has to be stored on the workstation is the TGS ticket and session key. However these have a lifetime limited by the lifetime granted to the TGS ticket. In fact, in the Athena Kerberos implementation all tickets obtained with the TGS are cached on the local workstation for efficiency reasons, however all expire at least when the TGS expires and are then useless. Therefore if a file of tickets is compromised the timeframe during which trouble may be directly caused is limited. More detail may be found in [2] and [3].

Multiple-Realm Support

The namespace that is represented by a set of Kerberos servers is referred to as a *realm*. In fact a fully qualified "Kerberos" name includes the name of the realm as a component. It is possible for the administrators of two realms to exchange a shared key between them that will then allow users from one realm to use services from the other realm without having to be registered in the other realm (of course services will know which realm a given user is from because the realm name is part of the fully specified Kerberos name). This feature facilitates the interoperation of applications across different organizations.

Programmers Viewpoint

MIT distributes an implementation of Kerberos free of charge in the U.S. This implementation includes a library of DES functions and a library of Kerberos functions.

Using this application library a programmer can typically add Kerberos authentication to a client–server application with a mere two library calls and some code to check return conditions. The library also allows an application programmer to securely exchange a session key between client application and server which may then be used by the applications to encrypt application specific data if desired.

Authentication vs. Authorization

It is important to understand that Kerberos is an *authentication* system, *not* an authorization system. Its mission is to prove the identity of (i.e., authenticate) a user requesting a service. It does not however provide any hint as to what service, if any, a given server should provide to a user. It is still the responsibility of each server to decide its own access control policy. This distinction between authentication and authorization is motivated by the fact that different services have different needs in terms of access control. For example, a fileserver may make access decisions based on the access control list of a given file. A print server on the other hand may want to associate a user with an internal accounting identity or credit card number and provide service if there is a positive balance or available credit to pay for the paper and toner consumed. However, both of these services need to be able to identify and authenticate the requester of the service.

Status of Kerberos

The Kerberos protocol specification is currently publicly available either via the Internet Drafts directory or directly from MIT via anonymous FTP from the host `athena-dist.mit.edu`. There are in fact two specifications, version 4, which for which source code is available within the United States (also distributed via `athena-dist.mit.edu`) and version 5 which is still under development. The code available for Kerberos includes the core Kerberos server, the necessary applications to fetch and manage the client ticket cache, A Kerberos application programmer interface library and a DES library. In addition versions of the Berkeley "R" commands that use Kerberos instead of trusted hosts is also provided. This code is written in **C** and is targeted for the UNIX environment but is portable to other environments.

Summary

The Kerberos Authentication system provides a means for a network user to prove its identity to a network service without any information going across the network in a fashion that allows its disclosure to compromise the authentication process.

References

[1] "Data Encryption Standard," National Bureau of Standards, FIPS Publication 46, Government Printing Office, Washington, D.C. (1977).

[2] "Section E.2.1: Kerberos Authentication and Authorization System," S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer, MIT Project Athena, Cambridge, MA (1987).

[3] "Kerberos: An Authentication Service for Open Network Systems," J.G. Steiner, B.C. Neuman, and J.I. Schiller, Proceedings Winter USENIX, Dallas Texas, February 1989.

JEFFREY I. SCHILLER received his S.B. in Electrical Engineering (1979) from MIT. As MIT Network Manager he has managed the MIT Campus Computer Network since its inception in 1984. Prior to his work in the Network Group he maintained MIT's Multics timesharing system during the timeframe of the ARPANET TCP/IP conversion. He is an author of MIT's Kerberos Authentication system. An active member of the IETF he chairs the Authentication Working Group [and enjoys many fine lunches and dinners...]

New Publication: *Journal of Internetworking—Research and Experience*

Background

Designing and implementing communication protocols that enable heterogeneous computer systems to interoperate is extremely difficult. The demand for such protocols has risen sharply as users have begun to connect diverse systems together.

Much of the recent work in internet protocols has been of an experimental or practical nature using the Internet as the testbed. As new international standards arise, there will be more implementation and experimental efforts. The results of this work have been reported mostly in technical reports, RFCs, and conference proceedings.

INTERNETWORKING will seek to provide a focus for original results of research in interoperability; a mechanism for quality control and verification through the refereeing process; and opportunities for comment and debate in a public and easily accessible forum. The journal will be published by John Wiley & Sons. The first two issues of *INTERNETWORKING* will be published in 1990.

Topics

The journal will encourage papers on topics ranging from architectural models to details of implementations and performance measurements. The emphasis will be primarily on those papers reporting practical experience gained in designing, implementing or using internet protocols. However, papers of a more mathematical or theoretical nature that might contribute to the development of, or better understanding of, methods of design, construction and use of internetworking protocols will also be welcomed. Shorter, unrefereed notes and letters may be published if they relate to current research topics or previously published articles. For further details please contact one of the editors.

INTERNETWORKING will provide a valuable source of reference to all who design, implement or use internet protocol software, or who want to learn more about the technology. Anyone involved in the investigation of interoperability or the communication protocols that support it should consider writing about their work for the journal.

Editor-in-chief

Professor Douglas E. Comer
Computer Sciences Department
Purdue University
West Lafayette, IN 47907
317-494-6009
comer@purdue.edu

Editors

Ralph A. Droms
Computer Science Dept
Bucknell University
Lewisburg, PA 17837
703-620-8990
rdroms@nri.reston.va.us

Deborah L. Estrin
Computer Science Dept
University of Southern California
Los Angeles, CA 90089-0782
213-743-7842
estrin@usc.edu

Liba Svobodova
IBM Research Division
Zurich Research Laboratory
8803 Ruschlikon
Switzerland
+41 1 724-8274
svo@ibm.com

Book Review

A must read

The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage by Clifford Stoll, Doubleday, 1989. ISBN: 0-385-24946-2. If you are interested enough in computer networks to read *ConneXions*, then you *must* read this book. It is absolutely essential reading for anyone that uses or operates any computer connected to the Internet or any other computer network. These may seem strong statements. They are, and they are true.

Don't worry, though, this is not a tough assignment. The book is very interesting, even exciting, to read. For many readers it will be a book you can't put down once you start. The book is so consistently interesting that you can open the book to any page at random and begin reading, and before finishing the page you will be hooked on the story.

This is the story of Cliff Stoll's fast track education as a UNIX system programmer and network wizard, with the main course being tracking down a spy! Cliff is really an astronomer, thrown into the computer support group at Lawrence Berkeley Laboratories by a temporary funding crunch on his astronomy project.

The 75 cent mystery

As his first task he looked into system accounting on the UNIX machines and found that two different accounting routines disagreed by 75 cents. While most of us would say that's close enough, Cliff decided to dig into it. And, one thing led to another.

It seemed that someone had set up a new account but had done only half the things needed to enter it into the accounting scheme. Looking into that account lead to the conclusion that there was a hacker in the system! The key question was: "Do we lock the hacker out or do we try to trap him?" The LBL decision was "trap him."

The story tells of the many ingenious schemes that Cliff used to track the spy, to keep him interested without letting him get too much critical data, or tipping him off to the fact he was being watched.

Security holes

The tricks the spy used are described, the data he was after (anything to do with SDI), his methods of access, of getting root (superuser) status, of cracking passwords. You must read about these and make sure the holes are fixed on the computer you use! The book names names, sites, computers, projects, accounts, passwords cracked. You may be *in* this book!

Dealing with the authorities and the carriers

It also describes the utter frustration of dealing with our government agencies (FBI, CIA, NSA) that are supposed to care about these problems. In contrast to this is the incredible response of the carriers (Tymnet, PacBell, AT&T, Bundespost) in tracing calls. There was an initial problem getting cooperation, but once the carriers were on board to do the tracing their cooperation and quickness were very impressive.

It is also a very human and personal story with clear characterizations of the people involved. It deals directly with Cliff's own life inside and outside the lab.

There is an epilogue on the Internet worm incident on November 1988. Summary: Read this book! *—Jon Postel*

CONNEXIONS
480 San Antonio Road
Suite 100
Mountain View, CA 94040
415-941-3399
FAX: 415-949-1779

FIRST CLASS MAIL
U.S. POSTAGE
PAID
SAN JOSE, CA
PERMIT NO. 1

CONNEXIONS

PUBLISHER	Daniel C. Lynch
EDITOR	Ole J. Jacobsen
EDITORIAL ADVISORY BOARD	Dr. Vinton G. Cerf, Vice President, National Research Initiatives.
	Dr. David D. Clark, The Internet Architect, Massachusetts Institute of Technology.
	Dr. David L. Mills, NSFnet Technical Advisor; Professor, University of Delaware.
	Dr. Jonathan B. Postel, Assistant Internet Architect, Internet Activities Board; Division Director, University of Southern California Information Sciences Institute.